Andrew Kozempel

Professor Avanzato

CMPSC 497

26 November 2023

**Lab #11: Deep Learning: Yolo Object Detection System**

**Objective:** To train the Yolo algorithm to identify and track crackers (1 class), analyze/optimize the algorithm for accuracy and efficiency, then validate the trained model with real world conditions using a webcam.

**Sample training images:**

| **Sample 1** | **Sample 2** | **Sample 3** | **Sample 4** | **Sample 5** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

**MATLAB Results:**

| **MATLAB Output** |
| --- |
| ans =  4×2 table  imageFilename cracker  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  {'C:\Users\Andrew\Documents\School\PSU\Fall 2023\497\Lab 11\good cracker\WIN\_20231102\_13\_14\_26\_Pro.jpg'} {[334 103 567 525]}  {'C:\Users\Andrew\Documents\School\PSU\Fall 2023\497\Lab 11\good cracker\WIN\_20231102\_13\_14\_31\_Pro.jpg'} {[ 375 57 625 605]}  {'C:\Users\Andrew\Documents\School\PSU\Fall 2023\497\Lab 11\good cracker\WIN\_20231102\_13\_14\_36\_Pro.jpg'} {[431 122 505 499]}  {'C:\Users\Andrew\Documents\School\PSU\Fall 2023\497\Lab 11\good cracker\WIN\_20231102\_13\_14\_41\_Pro.jpg'} {[447 132 496 473]}  numClasses =  1  anchorBoxes =  192 112  95 56  167 96  210 119  125 75  meanIoU =  0.8975  options =  TrainingOptionsSGDM with properties:  Momentum: 0.9000  InitialLearnRate: 1.0000e-04  LearnRateSchedule: 'none'  LearnRateDropFactor: 0.1000  LearnRateDropPeriod: 10  L2Regularization: 1.0000e-04  GradientThresholdMethod: 'l2norm'  GradientThreshold: Inf  MaxEpochs: 10  MiniBatchSize: 8  Verbose: 1  VerboseFrequency: 50  ValidationData: [1×1 matlab.io.datastore.TransformedDatastore]  ValidationFrequency: 50  ValidationPatience: Inf  Shuffle: 'once'  CheckpointPath: 'C:\Users\Andrew\AppData\Local\Temp\'  CheckpointFrequency: 1  CheckpointFrequencyUnit: 'epoch'  ExecutionEnvironment: 'auto'  WorkerLoad: []  OutputFcn: []  Plots: 'training-progress'  SequenceLength: 'longest'  SequencePaddingValue: 0  SequencePaddingDirection: 'right'  DispatchInBackground: 0  ResetInputNormalization: 1  BatchNormalizationStatistics: 'population'  OutputNetwork: 'last-iteration'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Training a YOLO v2 Object Detector for the following object classes:  \* cracker  Training on single CPU.  Initializing input data normalization.  |======================================================================================================================|  | Epoch | Iteration | Time Elapsed | Mini-batch | Validation | Mini-batch | Validation | Base Learning |  | | | (hh:mm:ss) | RMSE | RMSE | Loss | Loss | Rate |  |======================================================================================================================|  | 1 | 1 | 00:00:06 | 6.25 | 6.41 | 39.0586 | 41.1050 | 1.0000e-04 |  | 10 | 20 | 00:01:28 | 0.56 | 0.66 | 0.3152 | 0.4363 | 1.0000e-04 |  |======================================================================================================================|  Training finished: Max epochs completed.  Detector training complete.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

| **Output Figures** | | | |
| --- | --- | --- | --- |
|  |  |  |  |

| **Training Chart** |
| --- |
|  |

**Test Cases:**

| **Successes** | | | | |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

| **Failures** | | | | |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

**Conclusion:**

Overall, this was a straightforward lab that required small tweaks in the provided code. The model seemed to be pretty good using only 25 images, with a Mean Intersection of Union of 0.8975, which indicates that the model was able to align its predictions with the actual labels with decently high accuracy. The RMSE plot also shows a rapid decrease in a short amount of time, showing that the model was able to quickly learn and minimize errors between predictions and actual data.

When it came to actually testing out the data with a webcam, there were some issues. I had to lower the threshold a couple of times to get the test cases needed for the report. I started at 0.5, which was fine with detecting crackers that were very close to the camera. It seemed very ‘strict,’ so I had to lower it all the way to 0.25, which is pretty low. It worked for single crackers with higher thresholds, but 0.25 was basically the highest I could go if I wanted to detect more than 2 crackers in a single frame.

As for the failures, I think it could have been a mix of factors, including lighting. I think the main issue was that the training images were all very close and took up the majority of the photo, so that is why the cracker had to be close in real world testing. Also, they all had the same lighting and background. If I had to redo this, I would include varying angles, lighting, distances, and backgrounds.

I also included something that wasn’t a cracker for the failures. I am guessing that it just recognized a round object surrounded by a black background, so it recognized it as a cracker.

Overall, I think this lab was pretty successful, but it would have performed better if I had varying pictures like I mentioned.

**CODE:**

| **Yolov2wResNet50v3TrValTest.m** |
| --- |
| %% Test Yolov2 object detection (training, validation, test (no augmentation) version) % EDSGN 420 Spring 2021 RLA (last modified 4/20/2021) %  % This Yolov2 algorithm is based on MATLAB documentation... % https://www.mathworks.com/help//deeplearning/ug/object-detection-using-yolo-v2.html % % To start, load in gtruth groundtruth data from ImageLabeler or other % labeling tool into MATLAB workspace  if ~exist('gTruth') % exit script if gTruth not found  display('Must load gTruth variable into MATLAB workspace before execution')  return end  trainingDataTable = objectDetectorTrainingData(gTruth);   %% Display first few rows of the data set. trainingDataTable(1:4,:)  %% Split the dataset into training, validation, and test sets. Select 60% of the data for training, 10% for validation, and the rest for testing the trained detector.  rng(0); shuffledIndices = randperm(height(trainingDataTable)); idx = floor(0.8 \* length(shuffledIndices) ); trainingIdx = 1:idx; trainingDataTbl = trainingDataTable(shuffledIndices(trainingIdx),:);  validationIdx = idx+1 : idx + 1 + floor(0.1 \* length(shuffledIndices) ); % validationIdx = idx+1 : length(shuffledIndices); validationDataTbl = trainingDataTable(shuffledIndices(validationIdx),:);  testIdx = validationIdx(end)+1 : length(shuffledIndices); testDataTbl = trainingDataTable(shuffledIndices(testIdx),:);  % Use imageDatastore and boxLabelDatastore to create datastores for loading the image and label data during training and evaluation. imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});  % bldsTrain = boxLabelDatastore(trainingDataTbl(:,'redBall')); % labelName = char(trainingDataTable.Properties.VariableNames(2)) % label name is column 2 header in table % bldsTrain = boxLabelDatastore(trainingDataTbl(:, labelName)); % for % single label only  bldsTrain = boxLabelDatastore(trainingDataTbl(:, 2:end)); % for single or multiple labels  % The goal here is not to rely on using the literal/string name of the labels % (column labels 2 and beyond) in the data table.  imdsValidation = imageDatastore(validationDataTbl{:,'imageFilename'}); bldsValidation = boxLabelDatastore(validationDataTbl(:, 2:end));  imdsTest = imageDatastore(testDataTbl{:,'imageFilename'}); bldsTest = boxLabelDatastore(testDataTbl(:, 2:end));  %% Combine image and box label datastores. trainingData = combine(imdsTrain,bldsTrain); validationData = combine(imdsValidation,bldsValidation); testData = combine(imdsTest,bldsTest);  %% Display one of the training images and box labels. data = read(trainingData); I = data{1}; bbox = data{2}; annotatedImage = insertShape(I, 'Rectangle',bbox, 'Color', 'blue'); annotatedImage = imresize(annotatedImage,2); figure imshow(annotatedImage) reset(trainingData)  %% To reduce the computational cost of running the example, specify a network input size of [224 224 3], which is the minimum size required to run the network. inputSize = [224 224 3];  % Define the number of object classes to detect. numClasses = width(trainingDataTable)-1  %% Next, use estimateAnchorBoxes to estimate anchor boxes based on the size % of objects in the training data. To account for the resizing of the images  % prior to training, resize the training data for estimating anchor boxes.  % Use transform to preprocess the training data, then define the number of anchor boxes and estimate the anchor boxes. % Resize the training data to the input image size of the network using the supporting function preprocessData.  trainingDataForEstimation = transform(trainingData,@(data)preprocessData(data,inputSize)); numAnchors = 5; % use value of 5 and 7 typically [anchorBoxes, meanIoU] = estimateAnchorBoxes(trainingDataForEstimation, numAnchors)   %% Now, use resnet50 to load a pretrained ResNet-50 model.  featureExtractionNetwork = resnet50;  % Select 'activation\_40\_relu' as the feature extraction layer to replace the layers after 'activation\_40\_relu' with the detection subnetwork. This feature extraction layer outputs feature maps that are downsampled by a factor of 16.  % This amount of downsampling is a good trade-off between spatial resolution and the strength of the extracted features, as features extracted further down the network encode stronger image features at the cost of spatial resolution.  % Choosing the optimal feature extraction layer requires empirical analysis.  featureLayer = 'activation\_40\_relu';  %% Create the YOLO v2 object detection network.  lgraph = yolov2Layers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);  %% Use transform to augment the training data by randomly flipping the image and associated box labels horizontally.  % augmentedTrainingData = transform(trainingData,@augmentData);  % SKIP augmentation processing!!!!  augmentedTrainingData = trainingData;  % Visualize the augmented images. augmentedData = cell(4,1); for k = 1:4  data = read(augmentedTrainingData);  augmentedData{k} = insertShape(data{1},'Rectangle',data{2});  %reset(augmentedTrainingData); end figure montage(augmentedData,'BorderSize',10)   %% Preprocess the augmented training data, and the validation data to prepare for training.  preprocessedTrainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));  preprocessedValidationData = transform(validationData,@(data)preprocessData(data,inputSize));  %% Read the preprocessed training data. % calling read () below causes error % data = read(preprocessedTrainingData);  data = read(preprocessedTrainingData); % Display the image and bounding boxes. I = data{1}; bbox = data{2}; annotatedImage = insertShape(I,'Rectangle',bbox); annotatedImage = imresize(annotatedImage,2); figure imshow(annotatedImage)  data = read(preprocessedTrainingData); % Display the image and bounding boxes. I = data{1}; bbox = data{2}; annotatedImage = insertShape(I,'Rectangle',bbox); annotatedImage = imresize(annotatedImage,2); figure imshow(annotatedImage)   %% set training options  options = trainingOptions('sgdm', ...  'MiniBatchSize',8, ...  'InitialLearnRate',1e-4, ...  'MaxEpochs',10, ...  'CheckpointPath',tempdir, ...  'ValidationData',preprocessedValidationData, ...  'Plots','training-progress')      % Remove 'ValidationData',preprocessedValidationData, ... (RLA) %% train Yolov2 detector  [detector,info] = trainYOLOv2ObjectDetector(preprocessedTrainingData,lgraph,options);  % save the trained detector save('trainedDetector.mat', 'detector');  %% test with test training datastore removed (RLA) preprocessedTestData = transform(testData,@(data)preprocessData(data,inputSize));  % Run the detector on all the test images. detectionResults = detect(detector, preprocessedTestData);  % Evaluate the object detector using average precision metric. [ap,recall,precision] = evaluateDetectionPrecision(detectionResults, preprocessedTestData);  % The precision/recall (PR) curve highlights how precise a detector is at varying levels of recall. The ideal precision is 1 at all recall levels. % The use of more data can help improve the average precision but might require more training time. Plot the PR curve. figure plot(recall,precision) xlabel('Recall') ylabel('Precision') grid on title(sprintf('Average Precision = %.2f',ap))  %% function B = augmentData(A) % Apply random horizontal flipping, and random X/Y scaling. Boxes that get % scaled outside the bounds are clipped if the overlap is above 0.25. Also, % jitter image color.  B = cell(size(A));  I = A{1}; sz = size(I); if numel(sz)==3 && sz(3) == 3  I = jitterColorHSV(I,...  'Contrast',0.2,...  'Hue',0,...  'Saturation',0.1,...  'Brightness',0.2); end  % Randomly flip and scale image. tform = randomAffine2d('XReflection',true,'Scale',[1 1.1]); rout = affineOutputView(sz,tform,'BoundsStyle','CenterOutput'); B{1} = imwarp(I,tform,'OutputView',rout);  % Sanitize box data, if needed. A{2} = helperSanitizeBoxes(A{2}, sz);  % Apply same transform to boxes. [B{2},indices] = bboxwarp(A{2},tform,rout,'OverlapThreshold',0.25); B{3} = A{3}(indices);  % Return original data only when all boxes are removed by warping. if isempty(indices)  B = A; end end  function data = preprocessData(data,targetSize) % Resize image and bounding boxes to the targetSize. sz = size(data{1},[1 2]); scale = targetSize(1:2)./sz; data{1} = imresize(data{1},targetSize(1:2));  % Sanitize box data, if needed. data{2} = helperSanitizeBoxes(data{2},sz);  % Resize boxes to new image size. data{2} = bboxresize(data{2},scale); end  %helperSanitizeBoxes Sanitize box data. % This example helper is used to clean up invalid bounding box data. Boxes % with values <= 0 are removed and fractional values are rounded to % integers. % % If none of the boxes are valid, this function passes the data through to % enable downstream processing to issue proper errors.  % Copyright 2020 The Mathworks, Inc.  function boxes = helperSanitizeBoxes(boxes, imageSize) persistent hasInvalidBoxes valid = all(boxes > 0, 2); if any(valid)  if ~all(valid) && isempty(hasInvalidBoxes)  % Issue one-time warning about removing invalid boxes.  hasInvalidBoxes = true;  warning('Removing ground truth bouding box data with values <= 0.')  end  boxes = boxes(valid,:);  boxes = roundFractionalBoxes(boxes, imageSize); end  end  function boxes = roundFractionalBoxes(boxes, imageSize) % If fractional data is present, issue one-time warning and round data and % clip to image size. persistent hasIssuedWarning  allPixelCoordinates = isequal(floor(boxes), boxes); if ~allPixelCoordinates    if isempty(hasIssuedWarning)  hasIssuedWarning = true;  warning('Rounding ground truth bounding box data to integer values.')  end    boxes = round(boxes);  boxes(:,1:2) = max(boxes(:,1:2), 1);   boxes(:,3:4) = min(boxes(:,3:4), imageSize([2 1])); end end |

| **webcam1.m** |
| --- |
| load('trainedDetector.mat');  if ~exist('cam')  cam = webcam;  End  % connect to a webcam or ESP32-CAM while true  im = snapshot(cam); % if reading a jpg file, use im = imread('image1.jpg') and remove loop  im = imresize( im, [224, 224] );  [bboxes, scores, labels] = detect(detector, im, 'Threshold', 0.25); % apply the new network to the image  if ~isempty(bboxes) % if at least one object was detected....  im2 = insertObjectAnnotation(im, 'rectangle', bboxes, cellstr(labels), 'Color', 'red');     imshow(im2)  fprintf("The number of objects detected is %d\n", length(labels))  else  imshow(im)  end end % end while loop |